EXHIBIT A

Network Infrastructure Design

1 Overview

To bring intelligence to the telephony networks in a more rapid fashion, off of the switch service nodes (SN) were added to the network. These SNs were to have made it easier to add intelligence to the network by writing services that run on the SN. There are standards for the protocols that enabled SNs to interoperate, enabling distributed services to written on different providers' platforms. However, there is no standard for the application programming interfaces (APIs) that are used to write services. Thus, if a telephony service provider wanted to deploy a service on more than one platform in his network, he had to implement that service multiple times.

With the ongoing convergence of the PSTN, wireless, and Voice over IP (VoIP) telephony networks, this problem is even more acute. As people move from one network to another, they would like to do so seamlessly, without the loss of services. Service providers would like to provide all of their services on all of their networks. This becomes difficult and expensive as every service needs to be implemented several times for each network.

A goal of the Java Service Creation project is the creation of telephony and data services which are not connected to any particular network or any particular network hardware. The goal is a layered system in which different layers can be replaced to enable the overall platform to be matched with the real underlying networks as needed. However, the API exposed to the service writer would still be completely isolated from such changes below in the platform.

Stated in a slightly different fashion, this goal is the desire to merge the PSTN, wireless (cellular), and VoIP networks into one hybrid telephony network. This goal is apparent in several ways, including the desire to:

1. Have a single service platform for all of the networks.
2. To hide the network structure from services, enabling them to be, truly, network independent.
3. Enabling calls that span the networks.

The use of an abstract telephony API, such as the Java Telephony API (JTAPI), enables the goal of a single service platform for all of the networks, as the services are in no way tied to a specific network or protocol. The underlying infrastructure takes care of the necessary translation for the underlying network.

This document describes the design of the JTAPI implementation used to merge the telephony networks into one hybrid network. Also discussed, albeit at a high level, is the physical merging of the networks.

2 The Generic JTAPI Implementation

As with many other Java specifications, JTAPI comes in two parts. The first part is a set of Java Interfaces and Exception classes that are standard and can be gotten from Sun Microsystems' web site. The second part is a set of classes that implements the Java Interfaces found in the first part. This separation enables many different vendors to implement the JTAPI specification, while leaving applications independent of (and probably not even knowing) the names of the classes in any particular JTAPI implementation.

In general a JTAPI implementation looks like the picture in Figure 1 below. As described above, the application above the JTAPI API is completely independent of the JTAPI implementation below the JTAPI API.

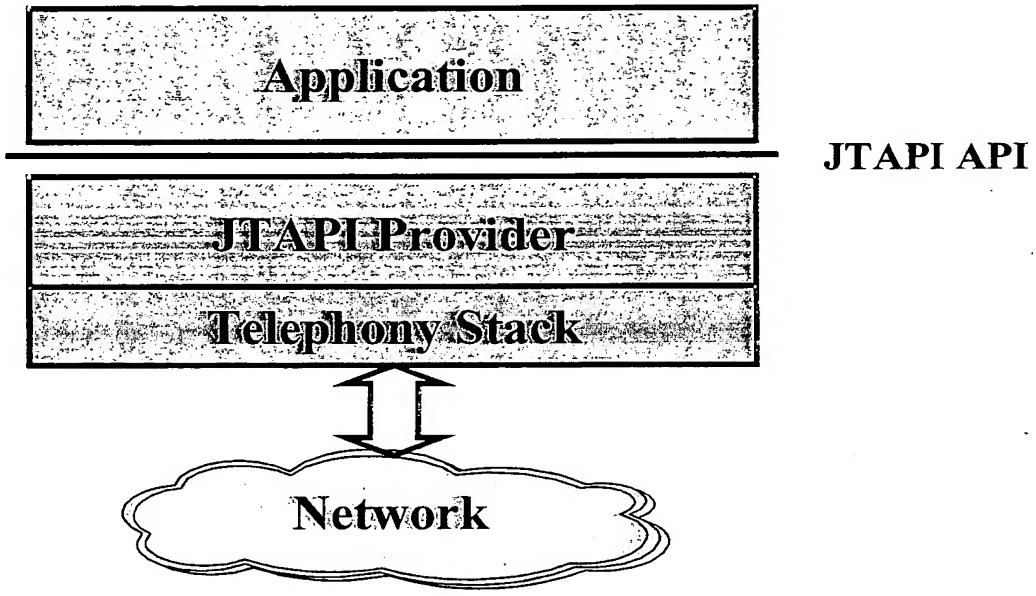

Figure 1. A typical JTAPI implementation.

While a JTAPI application is independent of the JTAPI implementation below it, a JTAPI implementation is typically tightly bound with the telephony stack below it. Portions of the JTAPI implementation need to map between JTAPI objects and methods and the telephony stack's interfaces.

Large portions of a JTAPI implementation are actually independent of the telephony stack used. Code that checks pre-conditions, deals with events, and does the base object manipulation is completely independent of the telephony stack. However, telephony stack

specific code needs to be sprinkled thru out this common code. Thus making it harder to share code between implementations.
The IBM Haifa Research Lab's Generic JTAPI Implementation was designed to overcome this specific problem, providing the following benefits:

1. Enabling JTAPI implementations for various telephony stacks and platforms to be written quickly.
2. The JTAPI specification is subject to much interpretation. All implementations based on the Generic JTAPI Implementation would have been written with a single interpretation.
3. Lower resource usage in cases where multiple telephony stacks are used simultaneously.

In the Generic JTAPI Implementation the JTAPI Provider is split into two layers:

1. The common JTAPI functions are in the Generic JTAPI layer.
2. The telephony stack specific function is in the Provider Plug-in layer.

As can be seen in Figure 2 below, the generic layer interfaces with the plug-in layer via the Java Telephony Service Provider Interface (JTSPI).

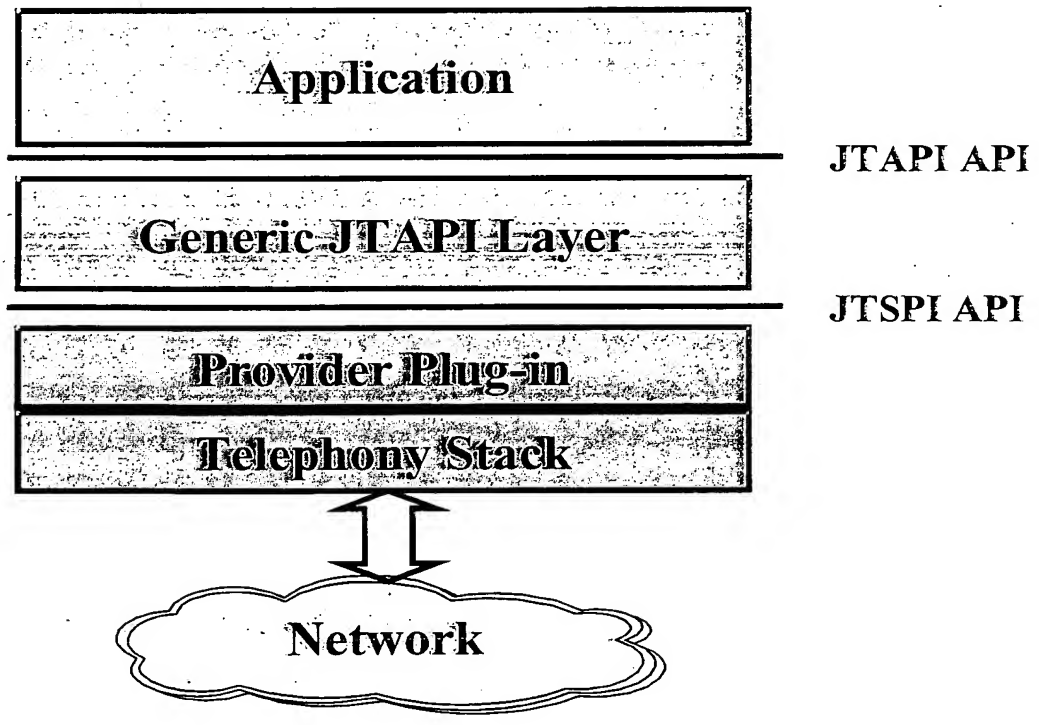

Figure 2.Generic JTAPI Implementation

Network Infrastructure Design

The Generic JTAPI layer dynamically loads Provider Plug-ins with the use of configuration data. Thus a "custom" JTAPI implementation tailored to the specific telephony stack being used can be built with ease.

Multiple Provider Plug-ins are supported simultaneously. In such cases they are assumed to be connected to different telephony networks thru the appropriate telephony stacks. Each plug-in would support it's own domain made up of a set of local and remote addresses as appropriate. However, from an application perspective, these networks can be viewed as one hybrid network. Thus meeting the project goal of a single telephony API over multiple networks on the same platform.

As the Generic Layer doesn't know anything about the configuration of the network below it, another component was added to the IBM Haifa JTAPI Implementation. This component is the Java Telephony Service Management Interface (JTSMI) plug-in.

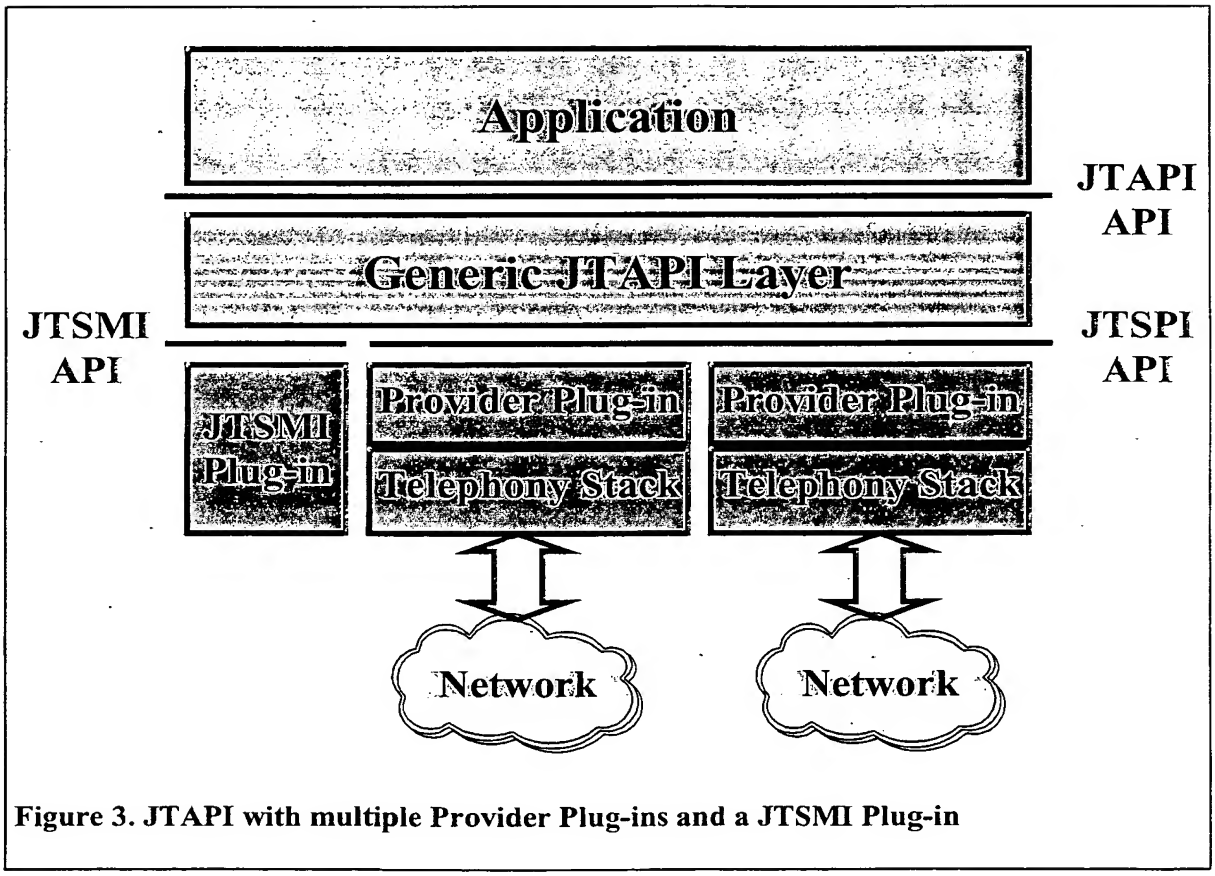

Figure 3. JTAPI with multiple Provider Plug-ins and a JTSMI Plug-in

As can be seen in Figure 3,above, the Generic JTAPI layer interfaces with the JTSMI plug-in thru the JTSMI API. The JTSMI plug-in's job is to select the appropriate provider plug-in to handle a particular call leg. Calls to the provider plug-in are actually made thru

the JTSMI plug-in. In fact, in some cases the JTSMI plug-in may issue different calls to the plug-ins than were made to it by the Generic JTAPI Layer. As such, the JTSMI plug-in is often tailored to the environment and understands the addressing conventions used in the applications running on top. It is planned that an external rules driven JTSMI plug-in will be written to avoid the need to constantly write new JTSMI plug-ins. For such a JTSMI plug-in new rules will be written to define the domains' of the Provider plug-ins in use.

The JTSMI plug-in also has knowledge of, at least at a high level, of the configuration of the network. Depending on the underlying network configuration, "hints" or "address prefixes" may be needed to tell the JTSMI plug-in where to direct the call leg.[1] Thought should be given to the use of the newly allocated ITU country code(s) for VoIP where appropriate.

[1] This is especially true if PSTN calls are being routed over VoIP networks in a Long Distance Bypass type of scenario.

3 Hybrid Network Calls

One of the main issues in determining exactly how calls that span networks work is the capabilities of the Gateway (GW) involved.

In general when bridging two networks both a Signaling GW and a Media GW are needed. The former translates signaling messages, while the later transcodes the media stream. While fundamentally there are two GWs involved in a call, often in fact, there is one GW that has combined the Signaling GW and the Media GW in one box. Most GWs built in this fashion expect to handle both the Signaling Translation and Media Transcoding. Efforts such as the IETF's MGCP and the ITU's Gateway Decomposition work are geared towards providing standard protocols to integrate decomposed GW. Such a decomposed GW is usually made up of:

1. A control element, called the Gateway Controller (GC).
2. A Signaling Gateway (SG) to translate signaling messages.
3. A Media Gateway (MG) to transcode the media streams.

Figure 4 on page 7 shows how a GC and a SG can be built from:

1. IBM HRL's Generic JTAPI.
2. Appropriate JTSPI plug-ins selected to support the protocols involved.
3. An appropriate JTSMI plug-in selected/written to manage the JTSPI plug-ins and the MG selected.
4. An application that redirects calls from one network's addresses to the other network's addresses[2].

[2] The application need not realize that it is in fact redirecting calls between the networks. There does need to be some sort of address translation/replacement to direct the JTSMI to create a hybrid network call.

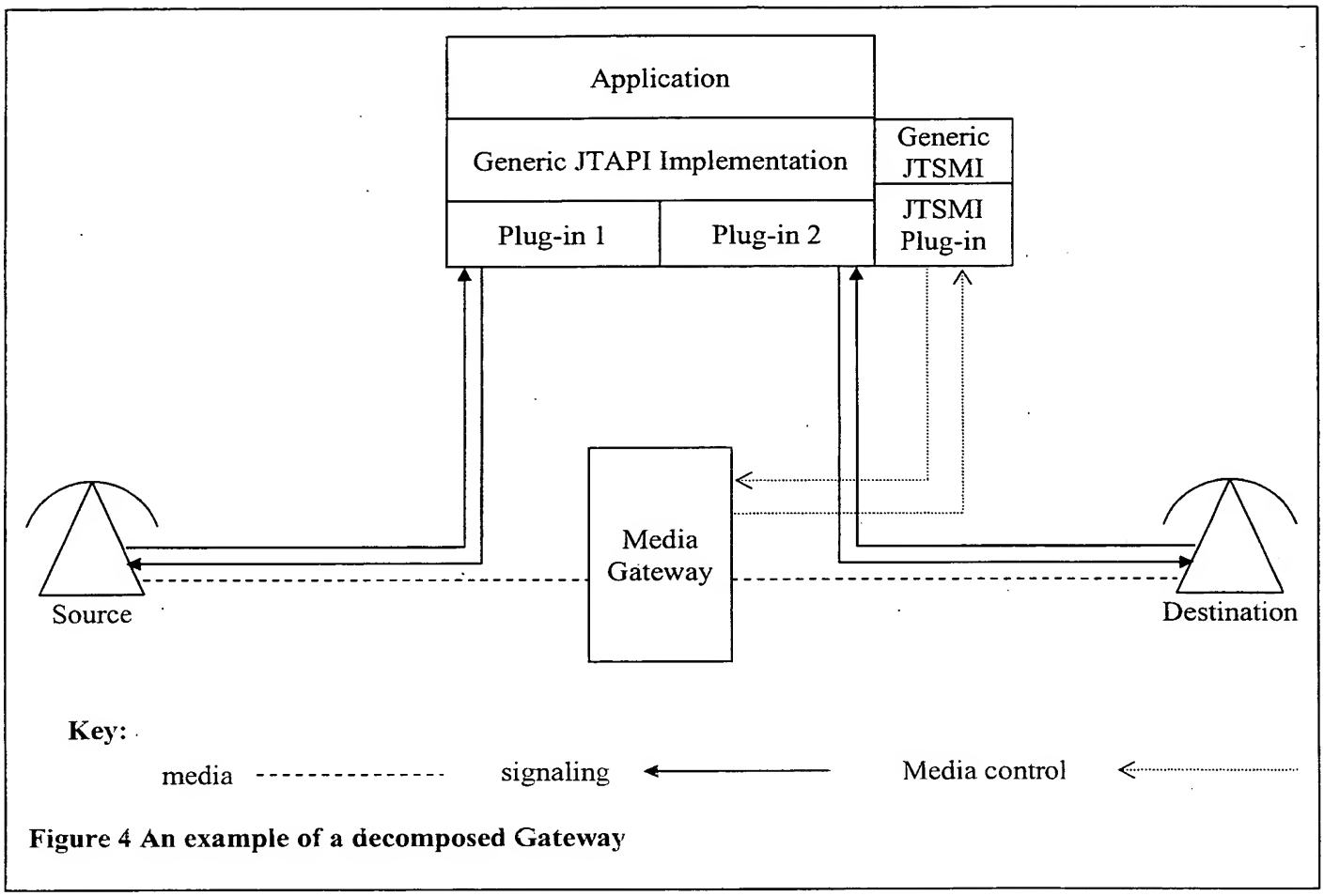

Figure 4 An example of a decomposed Gateway

In Figure 4, above, a third party calling model is shown. That is, a third party calling model from the perspective of the JTAPI application. However, from the perspective of the two JTSPI plug-ins involved, a pair of first party calls are placed. One call is an incoming call to Plug-in 1. The other call is an outgoing call from Plug-in 2. The Generic JTAPI combined with the JTSMI plug-in will make it look to the second plug-in as if the "second call" was originated by the real source of the hybrid call.

4 The Service Node

Services Nodes (SN) are made up of three main components:

1. A Service Execution Environment (SEE) component in which the services run.
2. A management component, which manages the SN.
3. A telephony network infrastructure component, which connects the SEE to the telephony networks below.

The design of the SEE and SN management component are discussed in their respective design documents.

The telephony network infrastructure component used in the Service Creation Project is IBM Haifa Research Lab's Generic JTAPI Implementation discussed in the section "The Generic JTAPI Implementation" on page 2.

From the perspective of the JTAPI implementation inside the SN the rest of the SEE is simply a JTAPI application. That is to say that it knows nothing special about the SEE.[3]

The exception to this is the JTSMI plug-in, which is in charge of associating call legs with the appropriate Provider plug-in. The JTSMI plug-in will need to understand the structure and addressing conventions of the networks below the JTAPI implementation. The use of an appropriate network-naming scheme along with an appropriate JTAPI implementation also enables the hiding of the underlying network's structure.

In particular, the JTSMI will need to handle PSTN (including both wired and wireless) and H.323 VoIP calls.

[3] It should be noted that in a SN with high availability characteristics there might be a need to have very tight integration between the rest of the SN and the JTAPI implementation. This is particularly true for the saving and restoring of the call state.

5 The Sonera Test Network

In the Sonera Test Network we will not have a decomposed GW. The RadVision GW is a monolithic GW that includes an SG and an MG under the covers. On the PSTN side it supports DTMF digit input using normal PSTN signaling. Calls can be routed thru it from PSTN to the H.323 network using single stage dialing. The GateKeeper (GK) will route H.323 calls that originate from the GW.

Based on this, all hybrid network calls originating from the PSTN must be routed to the H.323 network via a phone number. However, the H.323 terminal being called need not have registered itself with the GK with an E.164 alias. As a result this "phone number" must be:

1. Recognized by the PSTN and routed to the GW
2. Recognized by the GK and routed to the correct IP Terminal.

Our proposal is to make use of one of the new ITU-T E.164 country codes reserved for IP Telephony. All phone numbers with a specific country code and area code could be routed by the PSTN to our test network's GW. The rest of the phone number could either be:

1. The IP Terminal's IP address and port, represented numerically perhaps in the form aaabbbcccdddpppp.
2. A "cookie" used by the GK to recognize the desired IP alias to be translated.

In either case, neither the service nor the Generic JTAPI layer need know that such a "translation" is occurring. One of the JTSMI's jobs is to direct all calls from the Generic JTAPI layer to the correct JTSPI plug-in. When it does this it is capable of "translating" parameters passed. When it sees an IP alias used as a party address of a call that originated from the PSTN, it can replace the alias with a "phone number". The JTSMI plug-in could either:

1. Access the GK's Registration, Access, and Status (RAS) Data Base (DB) directly or via some side protocol.
2. Somehow register a numeric cookie for later use. The cookie would need to be destroyed after use as well.

In either case the GK's call routing code would need to recognize the special "phone number" and either use the IP address and port sent or lookup the signaling address via the specified cookie.

Figure 5 "Hybrid PSTN to H.323 call." on page 10, depicts a hybrid call originating in the PSTN and redirected to the H.323 network. A more detailed call flow can be seen in the section "Hybrid PSTN to H.323 Call Flow" on page 12.

Network Infrastructure Design

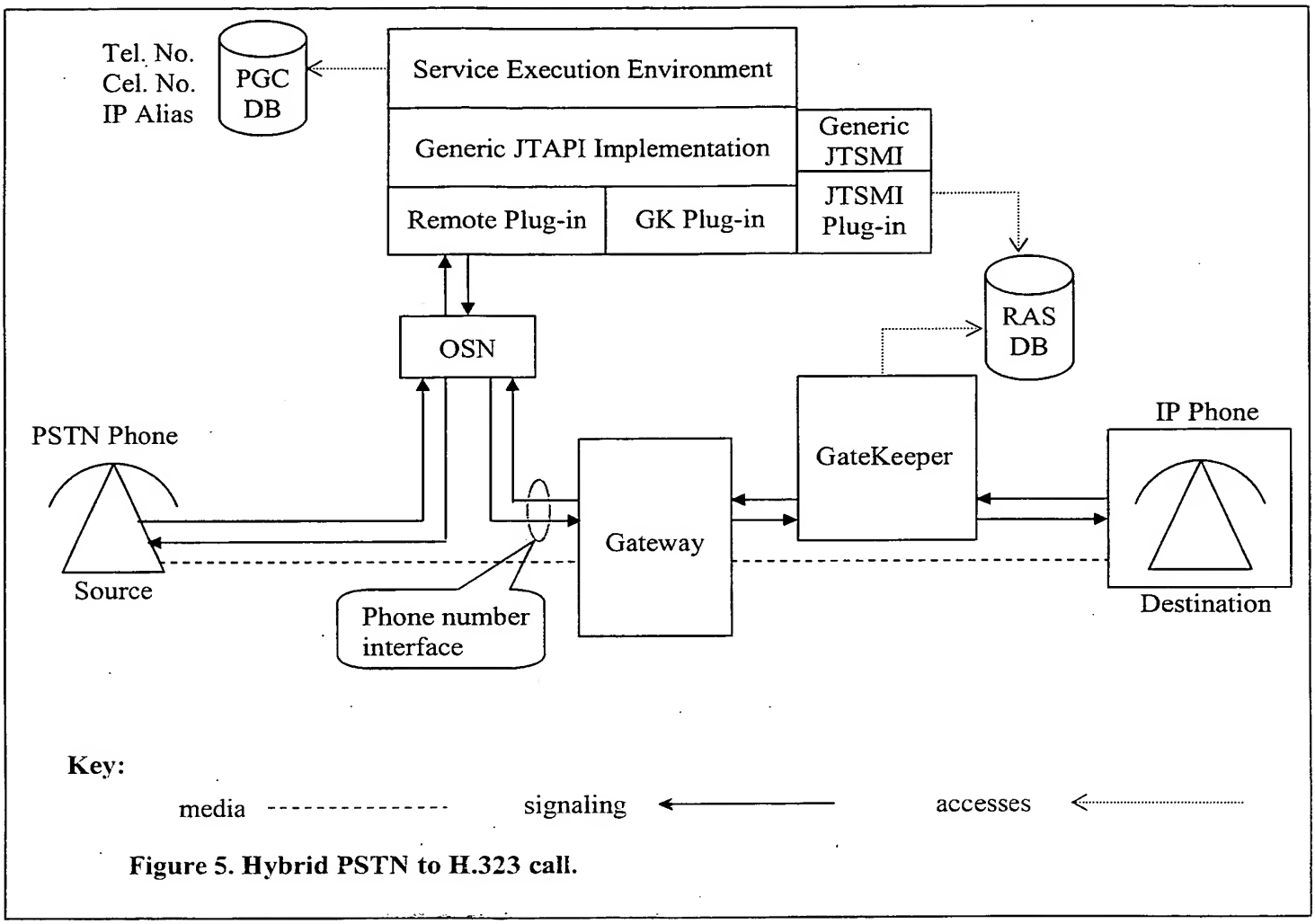

Figure 5. Hybrid PSTN to H.323 call.

While a hybrid PSTN to H.323 call is rather involved, a hybrid H.323 to PSTN all is much simpler. As the H.323 protocol supports connecting to an E.164 format phone number the call can be simply routed by the GK through the GW to the PSTN.

Figure 6 "Hybrid H.323 to PSTN call." on page 11, depicts a hybrid call that originated in the H.323 network and was redirected to the PSTN. A more detailed call flow can be seen in the section "Hybrid H.323 to PSTN Call Flow" on page 14.

Network Infrastructure Design

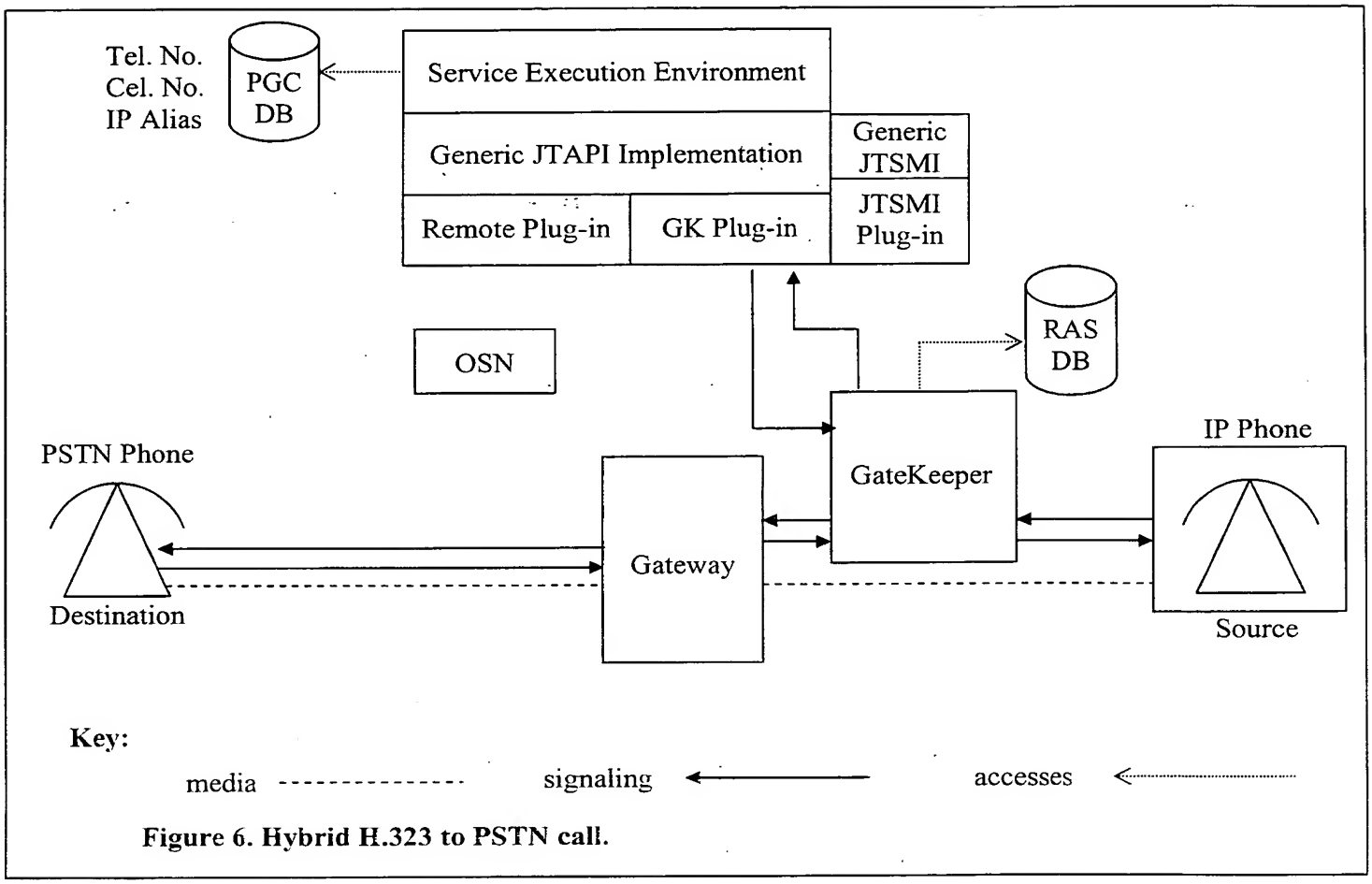

Figure 6. Hybrid H.323 to PSTN call.

6 Call Flows

6.1 Hybrid PSTN to H.323 Call Flow

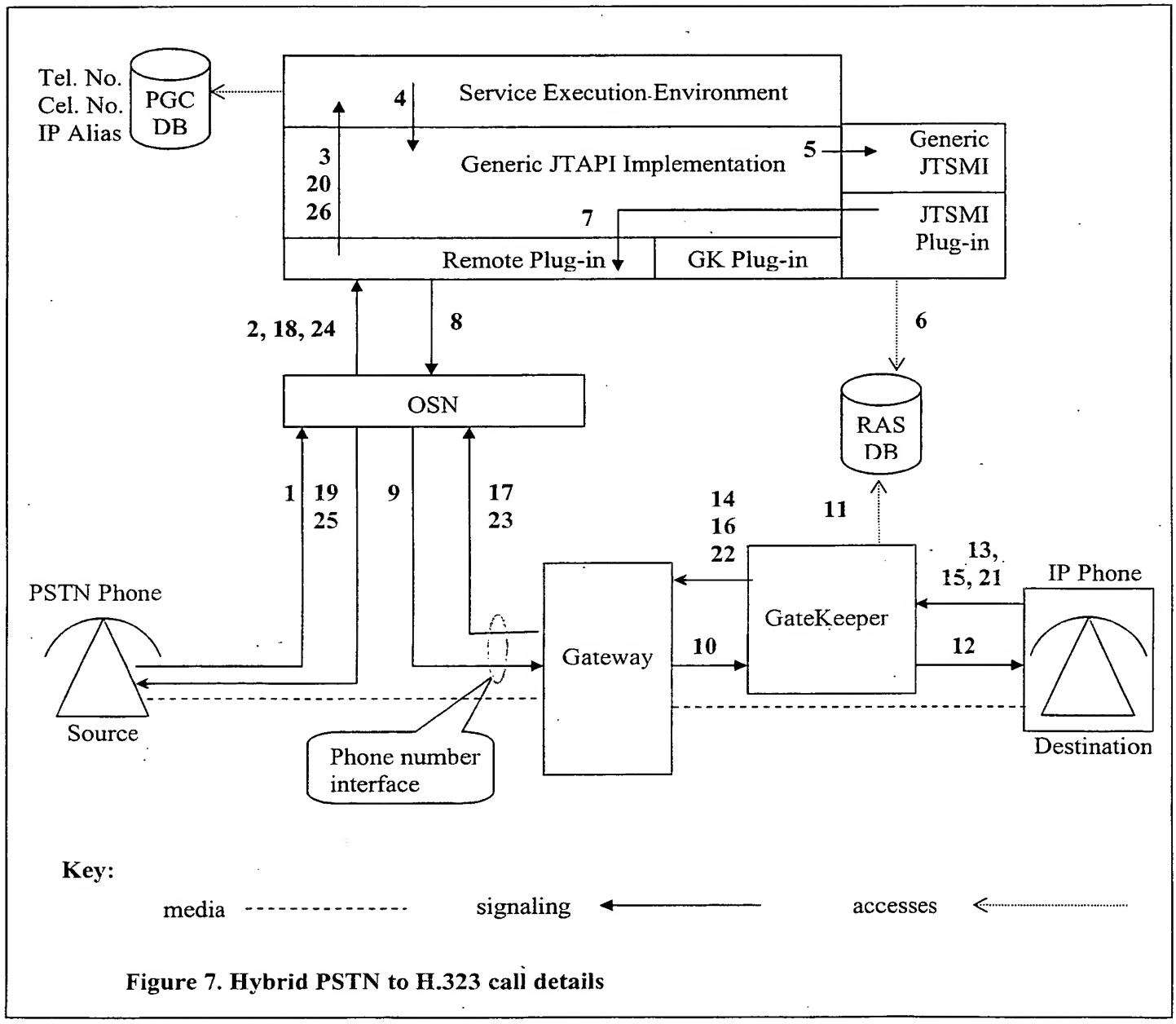

Figure 7. Hybrid PSTN to H.323 call details

6.1.1 Hybrid PSTN to H.323 Message Flow

Note: In the following description, SS7 messages are depicted by their three letter abbreviations in upper case (e.g. IAM). Messages of the Remote JTSPI protocol are depicted in italics (e.g. *Offering*). H.323 messages are depicted in all upper case (e.g. SETUP).

1. A call is placed on the PSTN and an IAM message is sent by PSTN to the Open Service Node (OSN).
2. An *Offering* message sent by the OSN to the Remote JTSPI Plug-in.
3. A CallCtlOfferedEv is sent to the SEE.
4. The service calls the redirect() method of the connection specifying an IP alias as the destination address.
5. The Generic JTAPI calls a method of the JTSMI to have the "redirect()" method of the Remote plug-in invoked.
6. The JTSMI plug-in recognizes a hybrid call that originates in the PSTN. It either looks up the IP alias in the RAS DB or registers a numeric cookie.
7. The "redirect()" method of the Remote JTSPI Plug-in is invoked with the special numeric destination address.
8. The *Redirect* message is sent to the OSN, again with the special numeric destination address.
9. The call is redirected by the PSTN to the GW.
10. The GW sends a SETUP message to the GK with a destination alias of the special numeric phone number.
11. The GK receives the SETUP message and, if necessary, looks up the cookie and translates the IP alias to the actual signaling address.
12. The GK sends the SETUP message to the destination IP Phone.
13. The IP Phone sends a PROCEEDING message to the GK.
14. The GK forwards the PROCEEDING message to the GW.
15. The IP Phone starts to ring and sends an ALERTING message to the GK.
16. The GK forwards the ALERTING message to the GW
17. The GW sends the OSN an ACM message
18. The OSN sends a *Ringing* message to the Remote JTSPI Plug-in.
19. The OSN forwards the ACM message to the PSTN.
20. CallCtlConnAlertingEv and CallCtlTermConnRingingEv events are sent to the SEE.
21. The IP Phone is answered and sends a CONNECTED message to the GK.
22. The GK forwards the CONNECTED message to the GW.
23. The GW sends an ANM message to the OSN.
24. The OSN sends a *Connected* message to the Remote JTSPI Plug-in.
25. The OSN forwards the ANM message to the PSTN.
26. CallCtlConnEstablishedEv and CallCtlTermConnTalkingEv events are sent to the SEE.

6.2 Hybrid H.323 to PSTN Call Flow

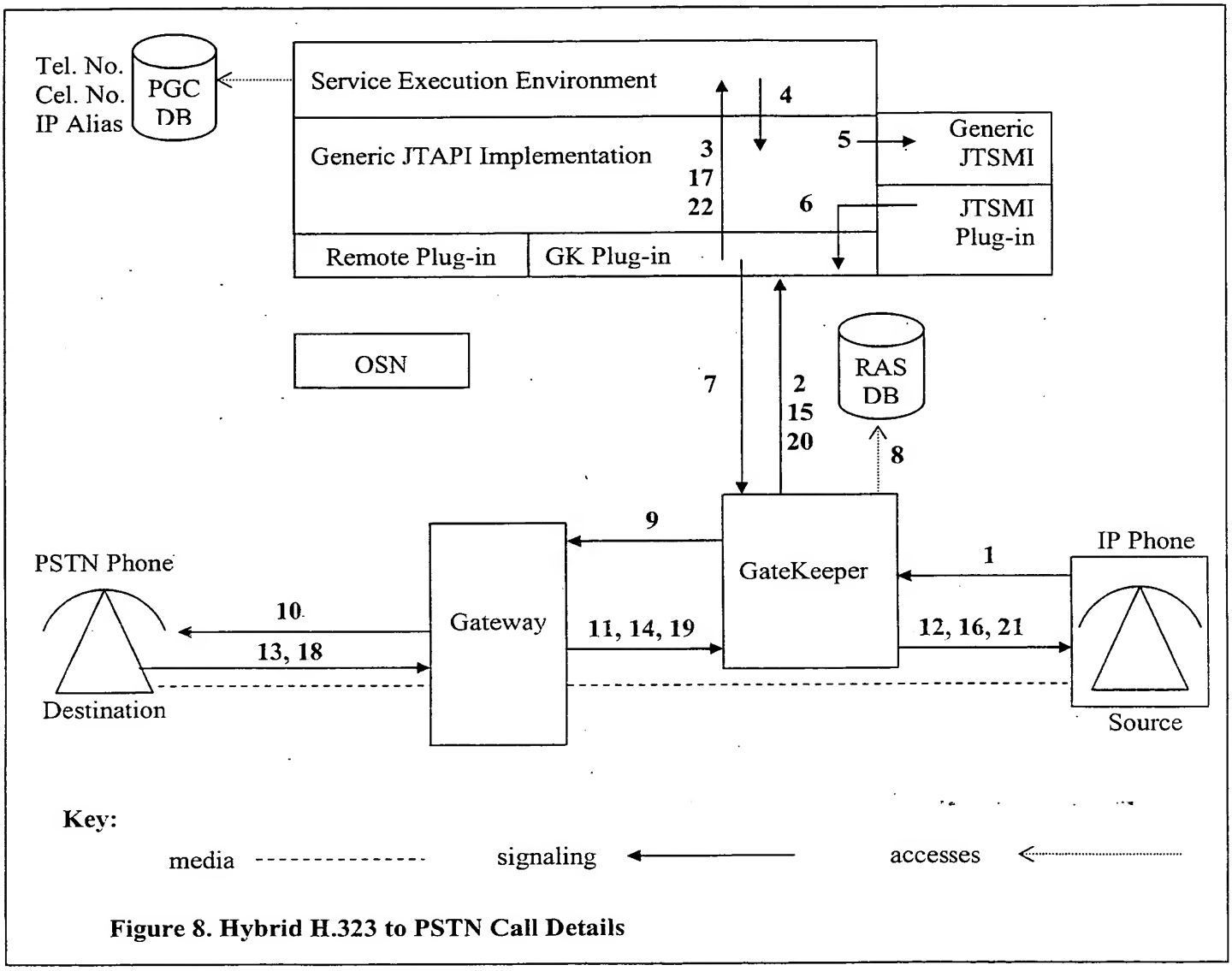

Figure 8. Hybrid H.323 to PSTN Call Details

6.2.1 Hybrid H.323 to PSTN Message Flow

Note: In the following description, SS7 messages are depicted by their three letter abbreviations in upper case (e.g. IAM). Messages of the Remote JTSPI protocol are depicted in italics (e.g. *Offering*). H.323 messages are depicted in all upper case (e.g. SETUP).

1. A call is placed on an IP Phone and a SETUP message is to the GK.
2. An *Offering* message sent by the GK to the GK's Remote JTSPI Plug-in.
3. A CallCtlOfferedEv is sent to the SEE.
4. The service calls the redirect() method of the connection specifying an PSTN phone number as the destination address.
5. The Generic JTAPI calls a method of the JTSMI to have the "redirect()" method of the Remote plug-in invoked.
6. The JTSMI plug-in recognizes a call that originates in the H.323 network and invokes the "redirect()" method call of the GK's Remote JTSPI Plug-in.
7. The *Redirect* message is sent to the GK, again with the address.
8. The GK looks up, if necessary the signaling address of the specified E.164 alias.
9. The GK sends a SETUP message to the GW with a destination address of the specified phone number.
10. The GW sends an IAM message to the PSTN.
11. The GW sends a PROCEEDING message to the GK.
12. The GK forwards the PROCEEDING message to the IP Phone.
13. The destination phone starts to ring and the PSTN sends an ACM message to the GW.
14. The GW sends an ALERTING message to the GK
15. The GK sends a *Ringing* message to the Remote JTSPI Plug-in
16. The GK forwards the ALERTING message to the IP Phone.
17. CallCtlConnAlertingEv and CallCtlTermConnRingingEv events are sent to the SEE.
18. The destination phone is answered and the PSTN sends an ANM message to the GW.
19. The GW sends a CONNECTED message to the GK.
20. The GK sends a *Connected* message to the Remote JTSPI Plug-in.
21. The GK forwards the CONNECTED message to the IP Phone.
22. CallCtlConnEstablishedEv and CallCtlTermConnTalkingEv events are sent to the SEE.

7 Issues

7.1 Future Work

1. Potential work, not within the scope of this project, would be to write H.248/MEGACO JTSPI and JTSMI plug-ins which could control external SG's and MG's. In such a configuration the H.248/MEGACO JTSPI and JTSPI plug-ins, the Generic JTAPI, and an application would simply be a GC.